Impossibility of Gathering, a Certification

Pierre Courtieu[2], Lionel Rieg[1,2], Sébastien Tixeuil[4,5], and Xavier Urbain[1,2,3]

[1] École Nat. Sup. d'Informatique pour l'Industrie et l'Entreprise (ENSIIE), Évry, F-91025
[2] CÉDRIC – Conservatoire national des arts et métiers, Paris, F-75141
[3] LRI, CNRS UMR 8623, Université Paris-Sud, Orsay, F-91405
[4] UPMC Sorbonne Universités
[5] Institut Universitaire de France

Abstract Recent advances in Distributed Computing highlight models and algorithms for autonomous swarms of mobile robots that self-organise and cooperate to solve global objectives. The overwhelming majority of works so far considers handmade algorithms and proofs of correctness.
This paper builds upon a previously proposed formal framework to certify the correctness of impossibility results regarding distributed algorithms that are dedicated to autonomous mobile robots evolving in a continuous space. As a case study, we consider the problem of gathering all robots at a particular location, not known beforehand. A fundamental (but not yet formally certified) result, due to Suzuki and Yamashita, states that this simple task is impossible for two robots executing deterministic code and initially located at distinct positions. Not only do we obtain a certified proof of the original impossibility result, we also get the more general impossibility of gathering with an even number of robots, when any two robots are possibly initially at the same exact location.

1 Introduction

The Distributed Computing community, motivated by the variety of tasks that can be performed by autonomous robots and their complexity, started recently to propose formal models for these systems, and to design and prove protocols in these models. The seminal paper by Suzuki & Yamashita [15] proposes a robot model, two execution models, and several algorithms (with associated correctness proofs) for gathering and scattering a set of robots. In their model, robots are identical and anonymous (they execute the same algorithm and they cannot be distinguished using their appearance), robots are oblivious (they have no memory of their past actions) and they have neither a common sense of direction, nor a common handedness (chirality). Furthermore, robots do not communicate in an explicit way. They have however the ability to sense the environment and see the position of the other robots. Also, robots execute three-phase cycles: *Look*, *Compute* and *Move*. During the *Look* phase, robots take a snapshot of the other robots' positions. The collected information is used in the *Compute* phase in which robots decide to move or to stay idle. In the *Move* phase, robots may

move to a new location computed in the previous phase. The two execution models are denoted (using recent taxonomy [8]) FSYNC, for fully synchronous, and SSYNC, for semi-synchronous. In the SSYNC model, an arbitrary non-empty subset of robots execute the three phases synchronously and atomically. In the FSYNC model, all robots execute the three phases synchronously.

One of the benchmarking [8] problems for mobile robots is that of *Gathering*. Regardless of their initial positions, robots have to move in such a way that they eventually stand on the same location, not known beforehand, and remain there thereafter. A key impossibility result for gathering is due to Suzuki & Yamashita [15]: two robots initially located at distinct positions may never gather if they execute a deterministic algorithm. This result is fundamental because any weakening of the initial system hypotheses (*e.g.* anonymity, obliviousness, common sense of direction) makes the problem solvable [5].

Related Works Most related to our concern are recent approaches to mechanising the algorithm design or the proof of correctness in the context of autonomous mobile robots [4,7,2,1]. Model-checking proved useful to find bugs in existing literature [2] and assess formally published algorithms [7,2], in a simpler setting where robots evolve in a *discrete space* where the number of possible positions is finite. However, no method exists to derive impossibility results using model checking. Automatic program synthesis (for the problem of perpetual exclusive exploration in a ring-shaped discrete space) is due to Bonnet *et al.* [4], and could be used to prove impossibility in a particular setting (by a side effect, if no algorithm can be generated), yet it exhibits important limitations for studying the gathering problem we focus on here. First, the authors consider only the discrete space setting (with a ring shape). Second, their approach is brute force (it generates every possible algorithm in a particular setting, regardless of the problem to solve). Third, the generator is limited to configurations where *(i)* a location can only host one robot (so, gathering cannot be expressed), and *(ii)* no symmetry appears (which eludes all interesting cases for studying gathering).

Developed for the Coq proof assistant[1], the Pactole framework enabled the use of high-order logic to certify impossibility results [1] for the problem of convergence: for any positive ε, robots are required to reach locations that are at most ε apart. Of course, an algorithm that solves gathering also solves convergence, but the converse is not true. As convergence is solvable in the usual setting, the impossibility results that can be obtained involve Byzantine robots (that is, robots that may exhibit arbitrary, and possibly malicious, behaviours). The impossibility results obtained in previous work using Coq [1] show that convergence is impossible if more than half of the robots are Byzantine in the

[1] `http://coq.inria.fr`

FSYNC model (or more that one third of the robots are Byzantine in the SSYNC model). These results cannot be directly extended to that of Gathering Impossibility for several reasons. First, they involve the active participation of Byzantine robots to destabilise the correct ones, while the gathering problem involves only correct robots. Second, the possible positions robots may occupy are encoded using rational numbers, while positions in the original model actually use real numbers.

Our Contribution In this paper, we consider the construction of a formal proof for the fundamental impossibility result of Suzuki and Yamashita [15], for two robots executing deterministic code and initially located at distinct positions. Our proof builds upon the previously initiated Pactole framework [1] to use actual real numbers as locations instead of rational numbers, and refines the definitions of executions (including scheduling assumptions) to enable the study of executions that involve only correct processes. Not only do we obtain a certified proof of the original impossibility result of Suzuki and Yamashita, we also get the more general impossibility result with an even number of robots, when any two robots are possibly initially at the same exact location.

2 Preliminaries

2.1 Certification and the Coq proof assistant

To certify results and to guarantee the soundness of theorems, we use the Coq proof assistant, a Curry-Howard based interactive prover enjoying a trustworthy kernel. The Pactole formal model is thus developed in Coq's formal language, a very expressive λ-calculus: the *Calculus of Inductive Constructions* (CIC) [6]. In this (functional) language, datatypes, objects, algorithms, theorems and proofs can be expressed in a unified way, as terms. λ-abstraction is denoted `fun x:T ⇒ t`, and application is denoted `t u`. Curry-Howard isomorphism associates proofs and programs, types and logical propositions. Writing a proof of a theorem in this setting amounts to building (interactively in most cases but with the help of tactics) a term the type of which corresponds to the theorem statement. As a term is indeed a *proof* of its type, ensuring the soundness of a proof thus simply consists in type-checking a λ-term.

Coq has already been successfully employed for various tasks such as the formalisation of programming language semantics [11,12] or mathematical developments as involved as the 4-colours [9] or Feit-Thompson [10] theorems.

The reader will find in [3] a very comprehensive overview and good practices with reference to Coq. Developing a proof in a proof assistant may nonetheless be tedious, or require expertise from the user. To make this task easier,

Pactole proposes a formal model, as well as lemmas and theorem, to specify and certify results about networks of autonomous mobile robots. It is designed to be robust and flexible enough to express most of the variety of assumptions in robots network, for example with reference to the considered space: discrete or continuous, bounded or unbounded...

We do not expect the reader to be an expert in COQ but of course the specification of a model for mobile robots in COQ requires some knowledge of the proof assistant. We want to emphasise that the framework eases the developer's task. The notations and definitions given hereafter should be read as the typed functional expressions they are.

The formal model we rely on, as introduced in [1], exceeds our needs with reference to Byzantine robots, which are irrelevant in the present work. Thus, for the sake of readability, a few notations have been slightly simplified: the pruned code essentially deals with taking into account the empty set of Byzantine robots in demonic actions. The reader is invited to check that the actual code is almost identical.

2.2 The Formal Model

The Pactole model[2] has been sketched in [1] to which we refer for further details; we recall here its main characteristics.

Two important features of COQ are used: a formalism of *higher-order*, which allows us to quantify over programs, demons, etc., and the possibility to define *inductive* and *coinductive* types [14], so as to express inductive and coinductive datatypes and properties. Coinductive types are in particular of invaluable help to express in a rather direct way infinite behaviours, infinite datatypes and properties on them, as we shall see with demons.

Robots are anonymous, however we need to identify some of them in the proofs. Thus, we consider given a finite set of *identifiers*, isomorphic to a segment of $\mathbb{N}$. We omit this set `G` (usually inferred by COQ) unless it is necessary to characterise the number of robots. If needed in the model, we can make sure that names are not used by the embedded algorithm.

Robots are distributed in space, at places called *locations*. We define a *position* as a *function* from a set of identifiers to the space of locations. The set of locations we consider here is the real line $\mathbb{R}$.

Robots compute their target position from the observed configuration of their siblings in the considered space. We also define permutations of robots, that is bijective applications from G to itself, usually denoted hereafter by Greek letters. Moreover, any correct robot is supposed to act as any other correct robot

[2] Available at `http://pactole.lri.fr`

in the same context, that is, with the same perception of the environment. For two real numbers $k \neq 0$ and t, a *similarity* is a function mapping a location x to $k \times (x - t)$, denoted $[\![k, t]\!]$. Real number k is called the homothetic factor, and $-k \times t$ is called the translation factor. Similarities can be extended to positions, by applying the similarity transform to the extracted location. This operation will be (abusively) written $[\![k, t]\!](\mathtt{p})$. Similarities are used as transformations of frames of reference.

For a robot *r-id*$_i$, a computation takes as an input an entire position $\mathtt{p}$ as seen by *r-id*$_i$, in its own frame of reference (scale, origin, etc.), and returns a real number l_i corresponding to a location (the *destination point*) in the same frame. As the robots are *oblivious* in the present context, the scale factor is taken anew at each cycle. Moreover to avoid any symmetry breaking mechanism based on identifiers, the result of r must be invariant by permutation of robots. We call *robograms* the embedded computation algorithms that fulfil this fundamental property.

Robograms may be naturally defined in a *completely abstract manner*, without any concrete code, in our COQ model as follows.

```
Record robogram := {
  algo : position → location ;
  AlgoMorph : ∀ p q σ, (q ≡ p ∘ σ⁻¹) → algo p = algo q }.
```

Demonic actions consist of a function associating to each correct robot a real number k such that $k = 0$ and the robot is not activated, or $k \neq 0$ and the robot is activated with a scale factor. An actual *demon* is simply an infinite sequence (stream) of demonic actions, that is a coinductive object.

```
Record demonic_action := {frame : G → R}.
CoInductive demon := NextDemon : demonic_action → demon → demon.
```

Characteristic properties of demons include *fairness* and synchronous aspects. We described in [1] how fair, FSYNC, and SSYNC demons could be defined using coinductive types. We show in Section 3 how k-fair demons can be expressed similarly.

Finally, an *execution* $(\mathtt{p}_i)_{i\in\mathbb{N}}$ from an initial position for (correct) robots $\mathtt{p}_0$ and a demon $(\mathtt{locate\_byz}_i, \mathtt{frame}_i)_{i\in\mathbb{N}}$, is an infinite sequence such that

$$\mathtt{p}_{i+1}(x) = \begin{cases} r_{[\![\mathtt{frame}_i(x), gp_i(x)]\!]}(\mathtt{p}_i) & \text{if } \mathtt{frame}_i(x) \neq 0 \\ \mathtt{p}_i(x) & \text{otherwise} \end{cases}$$

It is thus an object of type:

```
CoInductive execution :=
  NextExecution : (G → location) → execution → execution.
```

Its computation is reflected by a corecursive function `execute`.

3 Certification of Impossibility

The impossibility result we aim to prove formally is the following:

Theorem 1 *It is impossible to achieve the gathering of an even number of oblivious robots moving on the real line $\mathbb{R}$ with SSYNC k-fair demons for all $k \geq 1$.*

In this section, we specialise and enrich the Pactole model to provide a formal proof of this theorem. Note that for the sake of readability some notations may be slightly simplified compared to the actual code, available from `http://pactole.lri.fr`.

The main idea of the proof is taken from [15] while our premises are different: we allow for an unbounded number of robots, provided that it is even, and for an arbitrary initial position. On the contrary, [15] requires the initial position to have robots at distinct locations.[3]

To this goal: *(i)* we consider robots as points, that is two or more robots can occupy the same location, thus no constraint is added to the definition of a position, *(ii)* we assume robots enjoy strong global multiplicity detectors, the same global position is thus used for the computations of all robots, *(iii)* we consider that the travelling time is negligible, destination points returned by robograms are used directly to determine new locations, *(iv)* we consider oblivious robots, that is a new frame is chosen by the demon for each activation of any robot, *(v)* we take `location` to be `R`, the (axiomatic) definition of $\mathbb{R}$ in the Coq standard library `Reals`. Note that we are considering an unbounded continuous space.

3.1 *k*-Fairness

A demon is said to be *k-fair* if it is fair and k-bounded, that is such that between two successive activations of any robot, all other robots can be activated at most k times. Roughly speaking, k-fairness expresses the ratio between the most active robot and the less active one, as well as avoids the degenerated case of robots not being activated.

Firstly we express the property that, for any two robots `g` and `h`, the demon activates `g` within the k next activations of `h`. It consists in three cases of activation for an inital round. Either `g` is activated (its new frame is non-null) and we are done; this is case `kReset`, a base case. Either `g` is not activated but `h` is, and the property will holds for $k+1$ if it holds for k for the remainder of the demon (case `kReduce`). Finally if none of the two considered robots is activated during this round, the property holds for a certain k if it holds in the remainder of the

[3] This is why our results are not in contradiction with [15], Theorem 3.4, that exhibits a solution for a number of robots $n \geq 3$.

demon (case `kStall`) for the same k. Notice that if the latter case happens indefinitely, then one cannot prove `Between g h d` since `Between` is an *inductive* relation[4].

```
Inductive Between {G} g h (d : demon G) : nat → Prop :=
| kReset : ∀ k, frame (demon_head d) g ≠ 0 → Between g h d k
| kReduce : ∀ k, frame (demon_head d) g = 0
  → frame (demon_head d) h ≠ 0 → Between g h (demon_tail d) k
  → Between g h d (k + 1)
| kStall : ∀ k, frame (demon_head d) g = 0
  → frame (demon_head d) h = 0 → Between g h (demon_tail d) k
  → Between g h d k.
```

An infinite demon is thus k-fair, for a certain k, if `Between` holds for any couple of robots *at any time*, that is if the demon is k-fair (for the very same k) from the start and also for the remainder of the demon. We can express this coinductive property as follows.

```
CoInductive kFair {G} k (d : demon G) :=
  AlwayskFair : (∀ g h, Between g h d k) → kFair k (demon_tail d)
  → kFair k d.
```

Intended as a framework and a library, our formal development provides several theorems about k-fairness that may prove useful, namely that a k-fair demon is fair, that if a demon is k-fair, then it is k'-fair for all $k' \geq k$, etc.

3.2 Definition of Success

A robogram is a solution to the Gathering problem if robots reach the same, unknown beforehand, location within finite time regardless of their initial positions. First we define the property for a position `pos` of having all robots at a same location `pt`.

```
Definition stacked_at {G} (pos : G → location) (pt : location) :=
  ∀ r : G, pos r = pt.
```

Hence there is a gathering point for an execution at some step if for all future execution steps, the location is the same for all robots. Such an infinite behaviour is a coinductive property.

```
CoInductive Gather {G} (pt : location) (e : execution G) :=
 Gathering : stacked_at (execution_head e) pt
  → Gather pt (execution_tail e) → Gather pt e.
```

This situation has to occur *eventually*, which we thus define as an inductive property.

[4] The curly brackets around the first argument (`{G}`) set it as *implicit*, which allows us to omit it later on.

```
Inductive WillGather {G} (pt : location) (e : execution G) :=
  | Now : Gather pt e → WillGather pt e
  | Later : WillGather pt (execution_tail e) → WillGather pt e.
```

If this holds for a given robogram `r` and a given demon `d` from any initial position then `r` is a solution to the Gathering problem for `d`.

```
Definition solGathering {G} (r : robogram G) (d : demon G) :=
  ∀ (p : G → location),
   ∃ pt : location, WillGather pt (execute r d p).
```

We will prove that with a well chosen demon, even as constrained as a k-fair demon, there exists an execution where robots are always apart (we prove that this notion is in contradiction with being a solution). More precisely, there is an execution that keep half the robots away from the other half; that is: the position is split. In the following, `(G ⊎ G)` denotes the union of two disjoint sets isomorphic to the same segment of $\mathbb{N}$, hence guaranteeing an even number of robots. By construction, an element `g` of the left (respectively right) `G` is denoted `inl g` (respectively `inr g`). Moreover, recall that the location is obtained by application of the position (which is a function) to an identifier.

```
Definition Split {G} (p : (G ⊎ G) → R) :=
  ∀ x y : G, p (inl x) ≠ p (inr y).
```

The following coinductive property characterises such an execution:

```
CoInductive Always_Split {G} (e : execution (G ⊎ G)) :=
  CAS : Split (execution_head e)
   → Always_Split (execution_tail e)
   → Always_Split e.
```

In fact, the faulty execution we exhibit with this property in the proof leaves a particular position indefinitely *bivalent*: with the robots evenly distributed over two distinct locations only.

Of course, any execution for which this property holds cannot be compatible with a solution for a non-empty set of robots (of even cardinality).

```
Theorem Always_Split_no_gathering :
  ∀ (G : finite) (e : execution (G ⊎ G)),
   inhabited G → Always_Split e → ∀ pt, ¬ WillGather pt e.
```

3.3 The Theorem in Coq

We may now state a formal version of Theorem 1 as follows:

```
Theorem noGathering : ∀ (G : finite) (r : robogram (G ⊎ G)),
  inhabited G
  → ∀ k : nat, (1 <= k)
    → ¬ (∀ d, kFair k d → solGathering r d)
```

the proof of which amounts to showing that for a non-null even number of robots, any `k` and any robogram `r` there exists a k-fair demon that prevents r to gather all robots.

The proof we formalise is inspired from [15]; it makes use of two demons, one that is fully synchronous, and one that is 1-fair. Depending on the expected result of the first move, we use one or the other.

We consider an initial position consisting of two separate piles of the same number of robots. If the expected first move brings the robots of one pile onto the other pile, we choose the fully-synchronous demon, which results in switching the locations of the two piles, thus in obtaining an equivalent position. Otherwise, we choose the 1-fair demon that will activate only one pile at a time; the piles moving alternatively, a change of frame suffices then to get back to an equivalent position.

Both cases allow us to show that `Always_Split` holds, thus proving Theorem `noGathering`.

4 Remarks and Perspectives

Thanks to the abstraction level of the Pactole framework, setting the space to be $\mathbb{R}$, thus both unbounded and continuous, is not as complicated as one could imagine; it emphasises the relevance of a formal proof approach and how it is complementary to other formal verification techniques. In addition to the syntactical invocation of `R` and associated functions, the main change from previous formalisations (that in particular were dealing with $\mathbb{Q}$) addresses proofs more than specifications, and lies in the fact that we use axiomatic reals. With such a description of $\mathbb{R}$, there is no computation. Hence relations between two elements of type `R` must be actually proved as they usually cannot be obtained by computation primitives.

The size of the specialised development for the relevant notions and the aforementioned theorems (thus excluding for example the complete library for reals) is quite small, as it is approximately 480 lines of specifications and 430 lines of proofs. The file `noRDVevenR.v` itself is about 200 lines of specifications for 250 lines of proof scripts. This is a good indication on how adequate our framework is, as proofs are not too intricate and remain human readable.

References

1. Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium*

(SSS 2013), volume 8255 of *Lecture Notes in Computer Science*, pages 178–186, Osaka, Japan, November 2013. Springer-Verlag.

2. Béatrice Bérard, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. Formal verification of Mobile Robot Protocols. Technical report, May 2013.
3. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.
4. François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. Brief Announcement: Discovering and Assessing Fine-grained Metrics in Robot Networks Protocols. In Richa and Scheideler [13], pages 282–284.
5. Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012.
6. Thierry Coquand and Christine Paulin-Mohring. Inductively Defined Types. In Per Martin-Löf and Grigori Mints, editors, *International Conference on Computer Logic (Colog'88)*, volume 417 of *Lecture Notes in Computer Science*, pages 50–66. Springer-Verlag, 1990.
7. Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. Optimal Grid Exploration by Asynchronous Oblivious Robots. In Richa and Scheideler [13], pages 64–76.
8. Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
9. Georges Gonthier. Formal Proof—The Four-Color Theorem. In *Notices of the AMS*, volume 55, page 1370. december 2008.
10. Georges Gonthier. Engineering Mathematics: the Odd Order Theorem Proof. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 1–2. ACM, 2013.
11. Xavier Leroy. A Formally Verified Compiler Back-End. *Journal of Automated Reasoning*, 43(4):363–446, 2009.
12. John Mccarthy and James Painter. Correctness of a Compiler for Arithmetic Expressions. In *Proceedings of Applied Mathematica*, volume 19 of *Mathematical Aspects of Computer Science*, pages 33–41. American Mathematical Society, 1967.
13. Andréa W. Richa and Christian Scheideler, editors. *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium (SSS 2012)*, volume 7596 of *Lecture Notes in Computer Science*, Toronto, Canada, October 2012. Springer-Verlag.
14. Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.
15. Ichiro Suzuki and Masafumi Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.